(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 May 2001 (31.05.2001)

PCT

(10) International Publication Number
WO 01/39046 A1

(51) International Patent Classification[7]: G06F 17/30

(21) International Application Number: PCT/US00/32171

(22) International Filing Date:
22 November 2000 (22.11.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/167,327 24 November 1999 (24.11.1999) US

(71) Applicant *(for all designated States except US)*: OZ.COM [US/US]; 77 South Bedford Street, Burlington, MA 01803 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: PETURSSON, Hilmar [IS/IS]; Holtsbud 56, IS-56 Gardabaer (IS).

(74) Agent: STANIFORD, Geoffrey; Dergosits & Noah LLP, 4 Embarcadero Center, Suite 1150, San Francisco, CA 94111 (US).

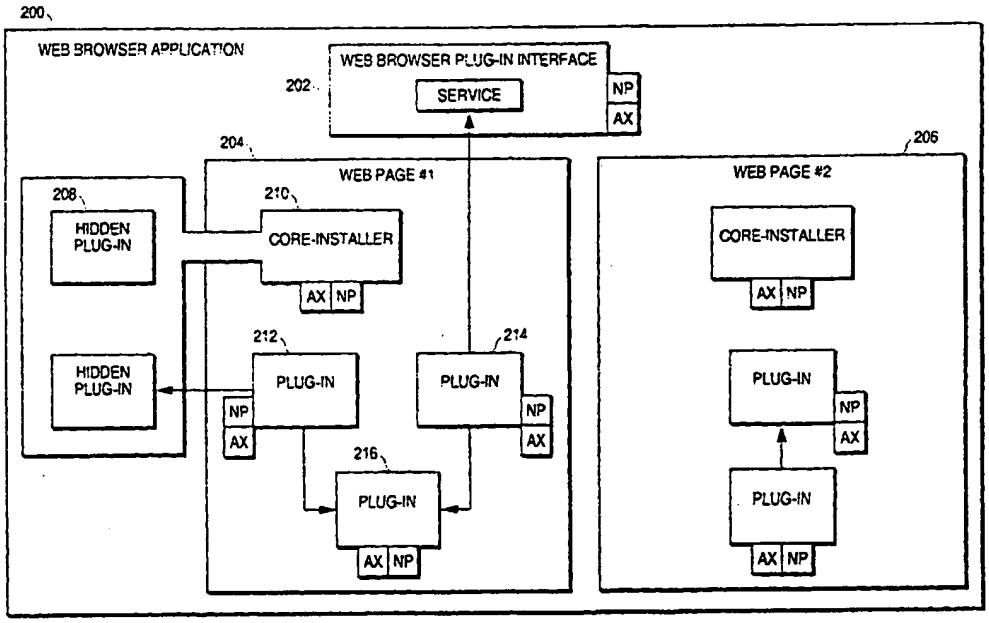
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

[Continued on next page]

(54) Title: WEB BROWSER PLUG-IN INTERFACE SYSTEM



(57) Abstract: A system for interfacing plug-in programs to a web browser is described. The web browser integration system comprises an application framework that is used to produce custom applications. Certain interfaces are defined that application components must implement in order to function within the interface framework. The web browser interface system contains a script engine that reads application scripts and automatically installs the components needed over the Internet. The interface system then plugs these components together according to instructions in the script and handles communication and interaction between them within the application. The plug-in integration framework is currently available as a plug-in program itself, or as a standalone application.

WO 01/39046 A1

— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WEB BROWSER PLUG-IN INTERFACE SYSTEM

FIELD OF THE INVENTION

The present invention relates generally to computer networks, and more specifically to a system for integrating and unifying plug-in programs for web browsers.

BACKGROUND OF THE INVENTION

Web browser programs are used by network client computers to access content on the World Wide Web portion of the Internet. A web browser communicates over the Internet with web server computers using the Hypertext Transfer Protocol (HTTP). The web browser program allows a user to access web pages on different server computers through hypertext links to these pages. The web browser program then formats and displays these pages on the client computer. Early web browser programs only allowed the access and display of simple web pages comprising text or simple graphical content. Present web browser programs, however, are substantially more powerful and allow access to many different types of content, such as complex graphics, streaming audio, streaming video, and other types of content. Moreover, the advent of browser based programming languages, such as Java® and ActiveX™, allow the incorporation of computer programs in web pages. This allows web pages to be dynamic and interactive, rather than simply a passive display application.

The functionality of a web browser is extended through the use of "plug-in" programs. A plug-in program is a computer program that integrates with the web browser and extends the capabilities of web browser program in a specific way. For example, plug-in programs are available that provide the ability to play audio samples or

view video movies from within a web browser. There are presently hundreds of plug-in programs available for the two most popular web browsers, Netscape Navigator™ and Microsoft Internet Explorer™, and several dedicated web sites have been created just to list and allow downloading of popular plug-in programs.

Plug-in programs allow a user to access a wide variety of browser enhancements by simply downloading the program and, in some cases, decompressing the program and/or running an installation script. Most current web browsers include facilities that manage and organize the various plug-in programs that a user has downloaded.

With the increasing development of Internet applications, different program formats and types of downloadable content are constantly being introduced. Application programmers must adapt to these changing programming formats to ensure that their plug-in programs are compatible with existing standards and protocols. One disadvantage of current web browser interface systems is that there is presently no mechanism to comprehensively manage custom applications. Instead, developers of the custom applications must tailor their plug-in applications to interface with the standards imposed by the web browsers. This limits the ability of developers to easily develop plug-in programs.

SUMMARY AND OBJECTS OF THE INVENTION

It is an object of embodiments of the present invention to provide a system that allows users to efficiently develop custom plug-in applications that share similar feature sets but have different user interfaces to allow access to and present the features.

It is a further object of embodiments of the present invention to allow custom applications to be deployed either as stand-alone applications or as plug-in programs to one or more different types of web browser programs.

A system for interfacing plug-in programs to a web browser is described. The web browser integration system comprises an application framework that is used to produce custom applications. Certain interfaces are defined that application components must implement in order to function within the interface framework. The web browser interface system contains a script engine that reads application scripts and automatically installs the components needed over the Internet. The interface system then plugs these components together according to instructions in the script and handles communication and interaction between them within the application. The plug-in integration framework is currently available as a plug-in program itself, or as a standalone application.

Other objects, features, and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description that follows below.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicates similar elements, and in which:

Figure 1 illustrates a computer network system that implements one or more embodiments of the present invention;

Figure 2 illustrates a functional block diagram of the plug-interface process within a web browser application environment, according to one embodiment of the present invention; and

Figure 3 illustrates how the web browser program communicates with the plug-in interface host through either the web browser interface function, according to one embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A system for interfacing plug-in programs to a web browser is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be evident, however, to one of ordinary skill in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form to facilitate explanation. The description of preferred embodiments is not intended to limit the scope of the claims appended hereto.

Hardware Overview

Aspects of the present invention may be implemented on one or more computers executing software instructions. According to one embodiment of the present invention, server and client computer systems transmit and receive data over a computer network or standard telephone line. The steps of accessing, downloading, and manipulating the data, as well as other aspects of the present invention are implemented by central processing units (CPU) in the server and client computers executing sequences of instructions stored in a memory. The memory may be a random access memory (RAM), read-only memory (ROM), a persistent store, such as a mass storage device, or any combination of these devices. Execution of the sequences of instructions causes the CPU to perform steps according to embodiments of the present invention.

The instructions may be loaded into the memory of the server or client computers from a storage device or from one or more other computer systems over a network connection. For example, a client computer may transmit a sequence of instructions to the server computer in response to a message transmitted to the client over a network by

the server. As the server receives the instructions over the network connection, it stores the instructions in memory. The server may store the instructions for later execution, or it may execute the instructions as they arrive over the network connection. In some cases, the downloaded instructions may be directly supported by the CPU. In other cases, the instructions may not be directly executable by the CPU, and may instead be executed by an interpreter that interprets the instructions. In other embodiments, hardwired circuitry may be used in place of, or in combination with, software instructions to implement the present invention. Thus, the present invention is not limited to any specific combination of hardware circuitry and software, nor to any particular source for the instructions executed by the server or client computers.

Figure 1 illustrates a computer network system 100 that implements one or more embodiments of the present invention. In system 100, a network server computer 104 is coupled, directly or indirectly, over line 125 to one or more network client computers 102 through a network 110. The network interface between server computer 104 and client computer 102 may also include one or more routers that serve to buffer and route the data transmitted between the server and client computers over line 121. Network 110 may be the Internet, a Wide Area Network (WAN), a Local Area Network (LAN), or any combination thereof.

In one embodiment of the present invention, the server computer 104 is a World-Wide Web (WWW) server that stores data in the form of 'web pages' and transmits these pages as Hypertext Markup Language (HTML) files over the Internet network 110 to the client computer 102. For this embodiment, the client computer 102 typically runs a "web browser" program 114 to access the web pages served by server computer 104 through a

web server process 116. In certain cases, web browser 114 can also be used to access data 108 served by other network servers, such as content provider 103. Examples of commercially available web browsers 114 that are executed by network client 102 include Internet Explorer™ by Microsoft® Corporation of Redmond, Washington and Netscape Navigator™ by Netscape Communications® of Mountain View, California.

In one embodiment of the present invention, client 102 executes a plug-in interface process 105 that represents an application framework that allows the development of custom applications and provides an interface through which native and non-native plug-in programs 127 can be managed and executed within the web browser program 114. The plug-in interface process 105 may represent one or more executable program modules that are stored within client computer 102 and executed by the client computer. Alternatively, however, it may be stored on a remote storage or processing device coupled to client 102 or network 110 and accessed by the client 102 to be locally executed. In a further alternative embodiment of the present invention, the plug-in interface process 105 may be implemented in a plurality of different program modules, each of which may be executed by two or more distributed server computers coupled to each other, or to network 110 separately.

In one embodiment of the present invention, wherein network 110 is the Internet, network server 104 executes a web server process 116 to provide HTML documents, typically in the form of web pages, to client computers coupled to network 110. To access the HTML files provided by server 104, client computer 102 runs a web client process (typically a web browser, such as Netscape Navigator™ or Microsoft Explorer™) 114 that accesses web pages available on server 104 and other Internet server

sites, such as content provider 103 (which may also be a network server executing a web server process). It should be noted that a network system 100 that implements embodiments of the present invention may include a larger number of interconnected client and server computers than shown in Figure 1. For this embodiment, the client computer 102 may access the Internet network 110 through an Internet Service Provider (ISP) 107.

As can be appreciated by those of ordinary skill in the art, the representative networked computers of Figure 1, such as network server computer 104 can be implemented as any standard computer that includes a central processing unit (CPU) coupled through a bus to various other devices. These devices could include random access memory (RAM), a read only memory (ROM), and mass storage devices (e.g., a magnetic disk, optical compact disk, or tape drive for storing data and instructions). The computer also typically includes input/output devices, such as, a display device, keyboard, and network interface device, along with other similar devices or interfaces. Any of the computers in Figure 1 could be implemented in the form of personal computers, laptop computers, mainframe computers, or other type of workstation computers. The computers in Figure 1 could also be implemented in the form of portable or miniaturized computing devices, such as handheld personal digital assistants (PDA), including devices that communicate with other devices on the network over a wireless medium. In certain systems, the client computer can also be implemented as a dedicated Internet client, such as a television that includes Internet access. Such clients are typified by the WebTV™ system.

Web Browser Plug-In Interface Process

In one embodiment, the web browser plug-in interface process 105 includes software components that are implemented to facilitate the creation and integration of various plug-in programs. The plug-in programs 127 may represent application programs that are written specifically for the plug-interface process ("native" plug-ins), and/or plug-in programs that are generic or not specifically written for the plug-in interface process ("non-native" plug-ins). The plug-in interface process 105 includes software components that allow non-native plug-ins to be adapted for use with the plug-in interface process.

The plug-in interface process is part of an overall web browser application environment that can be used to produce custom applications for use within web browser 114. The custom applications can be deployed either as standalone applications or web browser (e.g., Netscape Navigator or Internet Explorer) plug-ins. The plug-in interface process includes a script that describes several parameters of the custom application. The overall application environment consists of a set of plug-ins 127, an application script and the interface program available either as a standalone executable program, and/or as a web browser plug-in with an HTML page to define the application user interface. The plug-in interface process 105 is also able to install the custom application over the Internet and update it at the binary level with small update files called "patches".

The plug-in interface process 105 is divided into two parts. The first part is a single instance core, and the second part is a multiple instance plug-in host. The multiple instance plug-in host serves to host the plug-in programs and provide access to the features provided by the single instance core. The single instance core couples the plug-

in programs together according to parameters from an application script file and/or an HTML page and the execution context and hosting services.

From the point of view of the web browser, all plug-ins accessible through a web page are the essentially same plug-in program. Parameters from the page then specify which particular plug-in the interface should host within the space reserved by the HTML code for the web page. In this manner, several different plug-in programs can be connected or interfaced with one another to form a uniform application with respect to the web browser. The plug-in interface allows plug-ins to be mixed with bitmaps and other display entities and spread around a web page while still functioning logically together. In this manner, customized user interfaces can be quickly developed.

Figure 2 illustrates a functional block diagram of the plug-in interface process 105 within a web browser application environment, according to one embodiment of the present invention. The web browser application 200 represents a model in which a set of plug-in programs and services are connected together in a web browser page. A running instance of the web browser application is associated with a particular context. More than one instance of an application can exist in a single process. As shown in Figure 2, the web browser application 200 includes two individual web pages 204 and 206. Each web page represents a separate context of a running instance of the web browser program. An instance of an application is associated with a particular context. Thus, in Figure 2, web page 204 and web page 206 each depict separate and unique contexts of the plug-in interface process. Although Figure 2 illustrates an example in which the plug-in interface process is implemented as part of a web browser process, it should be

noted that the plug-in interface process could also be implemented as a stand-alone program executed on the client computer.

Each web page 204 and 206 include one or more plug-in programs. For example, web page 204 includes plug-in programs 212, 214, and 216. These can be native plug-ins that represent named objects. As such, they can receive parameters upon start up of the web page instance. A second type of plug-in is a hidden plug-in 208. This plug-in is not specified on a web page, but is created in a hidden window by core-installer 210 according to a parameter specified by the web browser plug-in interface 202. The core-installer 210 is a special purpose plug-in that installs and patches applications and instantiations of the plug-ins as needed.

The web browser plug-in interface 202 represents a single instance core of the interface process. The plug-in interface 202 includes one or more services, which are invisible static objects that communicate with the plug-ins for the web page contexts. The services do not receive parameters upon start-up.

The web browser application 200 creates all controls on a web page in an arbitrary order. Once the control has been loaded it has the option of reading parameters from the page before it gets the actual handle of the window that has been reserved for it. When a web browser opens a page containing what appears to be multiple instances of the plug-in interface controls, it starts loading them one by one.

When the web browser loads the first control, the single instance core of the plug-in interface 202 is created. Once the single instance core 202 has been created, all further creations of it will result in the same instance of the core, but not a new instance. This allows the plug-in interface to create a uniform application from multiple plug-ins.

Each plug-in entity within the web browser application 200 includes a browser interface that allows communication with the web browser program. For the example of Figure 2, the web browser interface for each plug-in is labeled "NP and AX" These represent communication means to the web browser though either an ActiveX (AX) interface in the case of Microsoft Internet Explorer, or the Netscape Plug-in Application Programming Interface (NP) in the case of Netscape Navigator.

Figure 3 illustrates how the web browser program communicates with the plug-in interface host through either the web browser interface function, according to one embodiment of the present invention. The plug-in interface host 306 acts as an interface between the browser and the plug-in 308 it is hosting. This enables a plug-in to function inside both types of browsers since they are not directly hosted by the browser, but rather inside an interface host that takes care of translating calls from the browser to the plug-in interface. Various web-page parameters 302 are passed to the plug-in 308 through the plug-in interface 306.

The plug-in host is created once the single instance core 202 has been created. As illustrated in Figure 3, the plug-in host reads three key parameters from the web page, these are labeled, "Name", "ClassID", "Connect". These parameters instruct the plug-interface process what control to put in the space reserved for a plug-in program.

Once the plug-in interface process has read the ClassID parameter, it requests the core-installer plug-in 210 to create an instance of a native plug-in (e.g., plug-in 212) with a plug-in Type ID equal to the ClassID. When the instance has been created, all other parameters are passed from the web page to the plug-in instance just created for reading. The plug-in interface process then looks at the Name and Connect parameters. The

Connect parameter (optional) contains one or more names of other plug-ins that this particular plug-in wishes to connect to. The plug-in host sends all connection requests to the single instance core. The single instance core builds a list of connection requests for each plug-in. It then monitors all plug-in creations by the Opium hosts and delegates the connection when plug-ins that have an outstanding connection request are created. All connections are made according to context. Thus, for example, if plug-in 214 wishes to connect to plug-in 216, it can only receive a connection to plug-in 216 within the same execution context. That is, the same execution context in a web page (e.g., web page 204) of the web browser. No connections between execution contexts are allowed.

During actual start-up and implementation of a plug-in interface process, particular complexities may be encountered regarding how to locate plug-ins on the client computer and creating an instance of them, installing them if they can not be found or upgrading them if they are outdated. All of these complexities are handled by core-installer special purpose plug-in. The core-installer plug-in resides in the same binary file as the plug-in interface process. This is ensures the availability of the core-installer plug-in. Each plug-in interface application must include the core-installer attributed according to its needs, thus, as shown in Figure 2, each web page 204 and 206 includes a core-installer plug-in. For a web page, this is done by placing the core-installer on the web page and giving it parameters specifying the application attributes. For a stand-alone application, this is done by containing it and attributing it in the standalone executable. The core-installer parameters that define an application are: Script, AppName, Version and View. AppName is the name of the application. The core-installer looks for a key by this name under a registry to find the location of the application content. The script

parameter specifies the URL (Uniform Resource Locator) to the application script. The version parameters specifies the lowest version of the AppName application needed to load the application. The View parameter specifies the name of a view section in the application script that defines what hidden plug-ins should be created by the core-installer. Hidden plug-ins have the access to the same features as visible plug-ins. The difference is that they are hosted by the core-installer in a hidden window instead of being hosted by a regular plug-in host.

The use of parameters transmitted from a web page to the plug-in interface allows for the automatic installation of plug-in and content according to instructions from an application script. This also allows the automatic upgrade of previously installed plug-ins and content as new versions become available. In one embodiment, plug-ins can be "torn" from a web page, by moving the plug-in from its embedded location on the web page to a separate window that can be moved around and resized. The plug-in interface process can also incorporate a file manager service that allows plug-ins to load application content with out having to know the physical location on the client computer. In addition, the interface process can include a persistent property manager that allows plug-ins to persist in a particular state in key-value pairs. The data can either be persisted on the client computer or on a separate persistence server.

Example Plug-In Application

The following description provides an example of how a simple application is built using the plug-in interface, according to one embodiment of the present invention. For this example, the application is a small chat application consisting of five separate plug-in programs, a logic plug-in, an input field, an output field, a user list plug-in, and a

sound engine plug-in. The application also uses four small sound files that are played on certain events in the chat application. To play these files, the chat application uses the sound engine plug-in. The application is deployed in a web browser so an HTML page is also provided.

The output field plug-in is a generic list control plug-in that accepts three parameters, direction, font, and color. The direction parameter specifies whether new lines to the history should be added from the bottom or the top. The font parameter specifies in what font lines in the history list should be rendered, and the color parameter specifies the color of the lists box background.

The input field plug is a generic input control that accepts two parameters, font and color. The font parameter specifies the font in which the input should be rendered, and the color parameter specifies the color of the input box background.

The logic plug-in contains the actual logic for the chat application. It handles server connection and the organization of users into chat groups. The logic plug-in accepts 3 parameters, login, password, and server. The login parameter is a string specifying the user's login ID. The password login is a string specifying the user's password, and server specifies the name of the server to connect to.

The user list plug-in displays a list of all the users that are currently chatting. This plug-in accepts no parameters.

The sound engine is a plug-in capable of playing sound files, it also takes care of mixing the different play requests that the plug-ins that connect to it make. The sound engine up-samples or down-samples the play requests when needed. This plug-in accepts three parameters, SamplesPerSec, channels, and BitsPerSecond. The SamplesPerSec

parameter specifies the bit rate at which the sound should be played. The channels parameter selects between mono or stereo mode, and the BitsPerSecond parameter specifies 8-bit or 16-bit resolution.

After the plug-ins and their associated parameters have been defined, an information file is compiled. The information file contains several sections in a file format denoted by a SectionName. Pseudo-Code for an information file for the example chat application is provided in Appendix A of this specification.

Once the information file is defined and processed, the HTML page for the application is defined. For the exemplary chat application, the user interface consists of two web pages. The first web page comprises a page that allows a user to enters his login and password. The second web page provides a page in which the user inputs and sees the chat session in progress. In one embodiment, because the login page needs to send parameters to the chat page, Microsoft Active Server pages are used to read the HTTP headers. However, this example could easily be converted to use regular CGI or some other web application model.

Microsoft Active Server Pages also provide the option of saving the redundancy of writing the plug-in definitions for both browser many times. Sample software pseudo-code that works in both Netscape and Microsoft browsers to insert a plug-in to a web page, according to one embodiment of the present invention, is provided as below. It should be noted that throughout the example pseudo code provided for the chat application, the generic web browser plug-in interface program may be referred to as "OPIUM", with appropriate labels associated therewith.

```
<object
        id          =       "OPIUM"
```

```
        classid         =       "CLSID:00000004-C001-FOOD-BABE-000BADCOFFEE"
        codebase        =       "http://www.oz.com/Opium/install/opium.cab"
        width           =       2
        height          =       1 >
        <param name = "OPIUM_Name"   value    =       "OPIUMServices">
        <param name = "OPIUM_ClassId" value    =       "{e05923b0-9cb8-11dl-a58c-
00a0c91e530e}">
        <param name = "Script"   value =
"http://www.oz.com/install/chat/b12/Chat.ozi">
        <param name = "Appname"         value = "Chat">
        <param name = "Version" value = "12">
        <param name = "View"    value = "Chat.view">
        <param name = "Logic.Login"     value = "Hilmar">
        <param name = "Logic.Password"  value = "1234">
        <embed
                type = "x-world/ozworld-4"
                width                   = 2
                height                  = 1
                align                   = "left"
                OPIUM_Name    = "OPIUMServices"
                OPIUM_ClassId = "{e05923b0-9cb8-11dl-a58c-00a0c91e530e}"
                Script                  = "http://www.oz.com/install/chat/b12/Chat.ozi"
                Appname                 = "Chat"
                Version         = "12"
                View                    = "Chat.view"
                "Logic.Login"           = "Hilmar"
                "Logic.Password"        = "1234"
                pluginpage              = "http://www.oz.com.Opium">
        </embed>
</object>
```

For the above example, some code is different between plug-ins and it needs to be entered twice for each plug-in. To eliminate this redundancy two ASP functions are used that can be included on a ASP page and used to embed controls. These are the functions AddParam and InsertControl. AddParam adds parameters to a global array called OPIUM_params. InsertControl adds a control to a page and gives it the parameters from the param array and clears the param array.

```
<%
Redim OPIUM_params (2,0)
Sub AddParam (name, value)
        top = unbound (OPIUM_params, 2)
        ReDim preserve OPIUM_params (2, top+1)
        OPIUM_params (0, top) = name
        OPIUM_params (1, top) = value
End Sub
```

```
Sub InsertControl (name, guid, width, height)
        code = VbCrLf
        code = code & "<object " & VbCrLf
        code = code & "  id        = ""'OPIUM'"" " & VbCrLf
        code = code & "  classid = ""'CLSID:00000004-C001-FOOD-BABE-000BADCOFFEE'"" " &
VbCrLf
        code = code & "  codebase = " " " " &
"http://www.oz.com/Opium/install/opium.cab" & " " " " & VbCrLf
        code = code & "  width = " & width & VbCrLf
        code = code & "  height = " & height & " > " & VbCrLf
        code = code & "  <param name = " "OPIUM_Name" "  value = " " " " & name & " " ">" &
VbCrLf
        code = code & "  <param name = " "OPIUM_ClassId" " value = " " " " & guid & " " ">" &
VbCrLf
        ArrayLimit = Ubound (OPIUM_params, 2) -1
        For I = 0 to ArrayLimit
        code = code & "  <param name = " " " " & OPIUM_params (0, I) & " " " value = " " " " &
OPIUM_params (1,I) & " " ">" & VbCrLf
        Next
        code = code & "  <embed" & VbCrLf
        code = code & "             type = " " x-world/ozworld-4" "  " & VbCrLf
        code = code & "             width    = " & width & VbCrLf
        code = code & "             height   = " & height &n VbCrLf
        code = code & "             align    = " "left" "  " & VbCrLf
        code = code & "             OPIUM_Name = " " " " & name & " " " " " & VbCrLf
        code = code & "             OPIUM_ClassId = " " " " & guid & " " " " " & VbCrLf
        For I = 0 to ArrayLimit
        code = code & "       " & OPIUM_params (0, I) & " =" " " " &
OPIUM_params (1, I) & " " " " "  & VbCrLf
        Next
        code = code & "   pluginpage   = " & "http://www.oz.com/Opium"
&" >" & VbCrLf
        code = code & "   </embed>"  &  VbCrLf
        code = code & "</object>" & VbCrLf
        Response.Write code
        Redim OPIUM_params (2,0)
End Sub
%>
```

By using this technique the HTML code above is reduced to the following ASP code:

```
<%
AddParam  "Script",              "http://www.oz.com/install/chat/b12/Chat.ozi"
AddParam  "Appname",             "Chat"
AddParam  "Version",             "12"
AddParam  "View",                "Chat.view"
AddParam  "Logic.Login",                 "Hilmar"
AddParam  "Logic.Password",      "1234"
InsertControl  "OPIUMServices", "{e05923b0-9cb8-11d1-a58c-00a0c91e530e} " , 2, 1
%>
```

The Login.html only contains one HTML form, as exemplified by the following pseudo-code:

```
<form method= "post" action = "chat.asp">
    <p>
        Login<br>
        <input type= "text" name= "login" maxlength="15" size="15">
    </p>
    <p>
        Password: <br>
        <input type="password" name="password" maxlength="15" size="15">
    </p>
    <p>
        <input type="submit" name="submit" value="Submit">
    </p>
</form>
```

The Chat.asp is the page where the user engages in the actual chat. Chat.asp reads parameters from the HTTP POST header from login.html and gives the controls parameters according to that. Pseudo-code for the chat example is provided below. Note that the code below specifies a page that does not include any aesthetics elements common to HTML programming.

```
<html>
<head>
<title>Chat</title>
</head>
<body>
<%
<%
AddParam "Script", "http://www.oz.com/Opium/install/chat/b12/Chat.ozi"
AddParam "Appname", "Chat"
AddParam "Version", "12"
AddParam "View", "Chat.view"
AddParam "Logic.Login", Request.Form ("Login")
AddParam "Logic.Password", Request.Form ("Password")
InsertControl "OPIUMServices" , "{e5923b0-9cb8-11d1-a58c-00a0c91e530e}" , 2, 1
%>
<!--
This asp code inserts the CoreInstaller plug-in. As you can see the plug-in is not called CoreInstaller but
OPIUMServices. This is done to give the idea that there is nothing magic about the names of the instance.
The instance name is used when reporting events from this plug-in to the user. All connections are also
done on the instance name. It is the GUID that gives the type of an Opium plug-in.
Parameters for the CoreInstaller:
```

"Script" here we give the location of the application script file that we are going to use. We have put our script file, called Chat.asp in the /Opium/install/chat/b12 directory of the www.oz.com server. We give the CoreInstaller plug-in this information with the "Script"parameter.
"Appname" tells the CoreInstaller plug-in that all plug-ins we want belong to an application named Chat.
"version" tells the CoreInstaller plug-in that the version of the application should be equal or greater than 12.
"view" tells the CoreInstaller plug-in that it should create all the hidden plug-ins in section Chat.view (that is where our logic plug-in and Sound Engine plug-in are listed).
Two parameters for hidden plug-ins are overridden here over the parameters from the application script. This is done by naming the hidden plug-in plus a '.' and then the parameter to override. Here we have overridden the "Login" and "Password" parameters for the logic object but keep the "Server" parameter as it is in the script.
- - >

```
<table border=1>
        <tr>
                <td>
                        <%
                        AddParam "Opium_Connect", "Logic"
                        InsertControl "UserList" , "{0DC81120-766A-618D-D000-00104BC64B92} ",
100, 200
                        %>
                        < ! - -
Here we insert the userlist plug-in. We connect it to the hidden logic
                plug-in created by the CoreInstaller, so the userlist can affect the state of the logic. The
                userlist plug-in accepts no other parameters.
                - ->
        </td>
        <td>
                <table>
                        <tr>
                                <td>
                                        <%
                                        AddParam "Direction", "bottom"
                                        AddParam "Font", "Helvetica, 14, 400,2"
                                        AddParam "Color" , "16777215"
                                        InsertControl "OutPut" , "{302CD780-B6E7-AA9A-F86D-
00104BC64B92}" , 250, 180
                                        %>
                                        < ! - -
                                        We don't need to connect the OutPut plug-in to
                                anything, the logic plugin connects to it from the application script file.
                                We just have to make sure that the name here is the same as the one the
                                logic plug-in is trying to connect to from the OZi file. The plug-in
                                specific parameters tell the output plug-in to have chat history to flow
                                bottom up, type of the font and the background color.
                                        - ->
                                </td>
                        </tr>
                        <tr>
                                <td>
                                        <%
                                        AddParam "Opium_Connect", "Logic"
                                        AddParam "Font", "Helvetica, 14, 400, 2"
                                        AddParam "Color", "16777215"
                                        InsertControl "Input", "{53F29A30-B6E7-AA9A-F86D-
00104BC64B92}", 250, 20
```

```
                                        %>
                                        <!--
                                        Here we connect the input plug-in to the logic so it
                                    can transmit text to the conference. The plug-in specific parameters tell
                                    the output plug-in the type of the font and the background color.
                                                -->
                                        </td>
                                    </tr>
                                </table>
                        </td>
                </tr>
        </table>
        </body>
        </html>
```

The next step is deployment. To deploy the chat application, three cabinets are created. These cabinets are as follows: chat.cab, stdplugs.cab and sound.cab. These are put in the information file, Chat.ozi.

If there will be an updated version, a patch is created, and then the version number of the web page is incremented. When the core-installer plug-in is initialized, it sees that the web page needs a newer version of the application than is currently installed and will try to download a patch and apply it.

A user wanting to use the chat application will open the login.html in the web browser. There he will enter his login and password and submit the form which gets posted to chat.asp. The asp engine executes the scripts specified in chat.asp and servers the HTML to the browser.

The browser parses the HTML given to it and for each reference embedding it, gives it the parameters. The object is always the same object, because of the single instance shared logic, and it builds up a list of all the plug-ins that are needed. When the plug-in interface process is requested to create the core-installer plug-in, it creates and gives it the list of the plug-ins that need to be created. Once the core-installer plug-in is

created it stops adding to the list of plug-ins to create and just forwards them to the core-installer as the requests are made.

When the core-installer plug-in is created, it reads in the AppName, Script, Version and View. It starts by checking if the application by the name. If the AppName is found, the core-installer searches for the FileRoot string value under that key and reads the path entered there. If all these checks are passed, the core-installer plug-in is ready to create all the plug-ins

In the foregoing, a system has been described for interfacing plug-in programs to a web browser. Although the present invention has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader spirit and scope of the invention as set forth in the claims. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

APPENDIX

```
[Application]
name = "Chat"
displayname = "Opium Chat"
version         = 12
views           = chat.view
```

The application section contains information about the application.

"name" is the name of the application, the CoreInstaller plug-in matches this name with the parameter from the web page and if a match occurs then it uses this script as an application script.

"displayname" This string is the human readable name of the application, it is displayed on all dialogs and messages that come from Opium. The string Opium never comes before the eyes of the user.

"version" is the application version. The CoreInstaller plug-in matches the parameter from the web page with this one. If the parameter on the web page is the same or lower the CoreInstaller plug-in knows that the application is up to date and it is safe to start creating plug-ins. If the version number given on the web page is higher a patch or a fresh install is needed. As we see here this is version 12 of our chat application.

"views" usually contains a list of fields that each specifies a name of a view section. In our case there is only one view available.

This is the view section specified above in the application section. It specifies what hidden plug-

ins and services should be available in this particular view.

"hidden" contains a list of fields that each specify a name of a section that defines attributes of the hidden plug-in. In our case we have two hidden plug-ins, the chat logic plug-in and the SoundEngine plug-in.

"services" contains a list of services that should be available to applications displaying this view of the application. In our case we only have one service, the FileManager that we need to allow plug-ins to locate the content (the sound files) on the users computer.

This is one of our hidden plug-in, the chat logic. Here all of its parameters are specified in the application script, this is unusual, normally parameters that are session oriented, like login and password come from the web page. These parameters are put here to show how parameters in the script are over ridded on the web page. Having these parameters here also gives the option of creating a simple web page allowing users to enter chat sessions on the default server "chat.oz.com" with out having to create a form for them to login. In that case all the user would bear the same name, "incognito".

"OPIUM_ClassId" the type identifier for the plug-in. It can not be overridden from the web page.

"OPIUM_Name" a human readable name for this particular instance of the OPIUM_ClassId plug-in. This name is for example used when connecting plug-ins together.

"OPIUM_Connect" can be overridden from the web page. Specifies a list of fields that each name a plug-in that this plug-in wishes to receive a pointer to, when each plug-in is created the single instance part of OPIUM checks if there is any plug-in that has asked for a connection to this newly created plug-in. If so the plug-in that asked for the connection will get a call to its

Connect function of IOZPlugin interface with an IUnknown pointer to the newly created plug-in. Then the ; plug-in wanting the connection usually queries the IUnknown pointer for an interface it knows how to communicate with.

Our logic object wants a pointer to the OutPut, the UserList and the SoundEngine so it can update these plug-ins as events in the conference occur. The logic plug-in does not connect to the input plug-in. The input connects to the logic because events flow from the input and to the logic not the other way around. Because of this it is simpler to have the input getting a pointer to the logic and query for a IAcceptText interface to feed text input to the logiv. In the case of the output and the userlist, it is simpler to have to logic receiving pointers to the plug-ins instead of vice versa (actually the userlist also connects to the logic because users can affect the sate of the logic through the userlist).

"Login" a plug-in specific a parameter, can be overridden from the web page.

"Password" a plug-in specific a parameter, can be overridden from the we page.

"Server" a plug-in specific a parameter, can be overridden from the web page.

This is the second one of our hidden plug-ins. It has the same OPIUM specific parameters as the logic above, with the exception of OPIUM_Connect which is missing because the Sound Engine is not connecting to any plug-in. The SoundEngine also has the plug-in specific parameters: SamplesPerSec, Channels and BitsPerSecond, explained above.

```
[SourceServerNames]
http://www.oz.com/Opium/install/Chat/b12

     This section contains a list of lines that each
     defines an URL where the installation image can be
     found.  If the CoreInstaller plug-in needs to do an
     install before, plug-ins can be created, it
loops
     through all of the lines until it finds the cabinet it
     is looking for.  In our case, there is only one URL so
     if the installation image can not
be
     found there  the installation will fail.

[PatchFiles]
http://www.oz.com/Opium/install/Chat/patches/12to13.rtp

     This section contains a list of lines that each
     defines an URL to a patch file.  If the CoreInstaller
     plug-in needs to do an
upgrade
     to the application before it can start creating plug-
     ins it uses these URL to find a patch file to update
     the application.  In our case we
have  ; only one possible patch file location, if the patch
is not found
there
     the only resort is a complete install from the
     locations specified in
the
     [SourceSereverNames] section.


     DefaultInstall is an install section.  Install
     sections contain information about the nature of an
     installation.  An application script can specify many
     install section that each have different instructions
     in the.  The install sections are referred to by a
     parameter to the CoreInstaller.  The Name
     DefaultInstall is a special one because that does not
     need to be referred to from the web page.  If no
     section is specified, DefaultInstall is used.

     "RequiredDiskSpace" defines the amount of free disk
     space the drive that the users selects to install to,
     needs to contain to make the installation.  This is
     the work space the installation needs, not necessarily
```

the amount the application will take after the installation is complete. That is usually lower than the work space needed as cabinets can be deleted after they have been extracted.

"CopyFiles" contains a list of fields that each is a name of a copy-file-section. The CoreInstaller plug-in downloads and installs all the sections that are listed here. In our case we ask that two copy-file-sections be installed, Chat.app and Chat.content.

"AddReg" contains a list of field that each is a name of an add-reg-section. The CoreInstaller plug-in will add all lines specified in the add-reg-sections to the registry. In the case of our Chat application, we ask that only the Chat.reg section be added to the registry.

Chat.app is a copy-file-section as we see in the DefaultInstall section above. Copy-file-sections contain a list of lines that each have two fields, the name of the cabinet and its estimated size. The CoreInstaller plug-in tries to download each cabinet from any of the URLs from the SourceServerNames section. When all the cabinets have been downloaded, the CoreInstaller plug-in extracts them to the locations ; specified by the DestinationDirs section.

This copy-file-section lists two cabinets to install.

"chat.cab" here we put CLogic.dll file that contains the logic plug-in and the user list plug-in.

"StdPlugs.cab" here we put stdplugs.dll that contains, among others, the input plug-in and the output plug-in. We also put the SndEng.dll in this cabinet.

Actually we could just have one cabinet here and put both dlls into that cabinet. We put our dlls in

different cabinets to better show possibilities of a copy file section.

This is another copy-file-section, it contains only one cabinet, the sound cabinet that contains the sounds that the chat application will play on several occasions.

Chat.reg is an add-reg-section as we see it is referred to by the DefaultInstall section above. Add-reg-sections contain a list of lines that the CoreInstaller plug-in should add to the registry. Each line in the list consist of 5 fields, they are:

0. specifies under what root key the entry should be added, possible values are: "HKCR", "HKCU", "HKLM" and HKU.

1. specifies the sub key where the entry should be added. Escape sequences are need to for backslashes, because values with white spaces in them are not possible in the OZi script format unless they are string (values inside " "). Strings in the OZi file format follow the same files a C strings.

2. the value to be created under the subkey.

3. the type of the data that follows, possible values are: "REG_SZ" and "REG_DWORD".

4. the data to enter.

Our chat application enters nothing to the registry. The line above is a comment and is only kept here as an example of how the CoreInstaller plug-in can be made to update the registry.

The DestinationDirs section contains information about the destination

that each cabinet in a copy-file-section should be extracted to. It consists of a list of lines that bear the name of their respective copy-file-sections. The first field in each line is a number, each number has a special meaning to the CoreInstaller plug-in:

1 is the application directory as selected by the user.

10 is the windows directory on the computer.

11 is the windows system directory on the computer.

The second field contains a relative path under the destination directory.

In both the copy-file-sections in our chat application are to be extracted to the user selected application directory. In addition to that we say that all cabinets in the Chat.content section should go in a subdirectory. sound under the user selected directory. We could have also done this by specifying relative information in the sound.cab cabinet but this gives a better idea of the capabilities of the CoreInstaller plug-in.

CLAIMS

What is claimed is:

1. A method of interfacing plug-in programs within a web page, comprising the steps of:

identifying one or more plug-in programs to be executed within the web page;

defining a script format defining a plurality of parameters included within each plug-in program;

providing a plug-in interface program that unifies the plurality of parameters for each plug-in program, resolves dependencies among the one or more plug-in programs, and establishes the display and execution of each plug-in program within the web page.

2. The method of claim 1 wherein the web page is displayed on a client computer through a web browser application locally executed on the client computer;

3. The method of claim 2 wherein the web browser program comprises one of Netscape Navigator and Internet Explorer, and wherein each plug-in program includes an interface allowing the plurality of parameters for each plug-in program to be passed to the web browser program.

4. The method of claim 3 wherein the plurality of parameters comprises a plug-in name, a plug-in identifier, and a plug-in connect parameter.

5. The method of claim 4 wherein the plug-in connect parameter defines a name of one or more plug-in programs that a first plug-in program will connect to.

6. The method of claim 4 wherein a subset of the one or more plug-in programs are native plug-in programs written for the plug-in interface program.

7. A client computer coupled to a server computer over a network, the client computer comprising:

a web browser process accessing World Wide Web content in Hypertext Markup Language format from a web server process executed on the server computer;

one or more plug-in processes executed in conjunction with the web browser process and operable to display content within a web page displayed on the client computer;

a plug-in interface process executed in conjunction with the web browser process and configured to define a common set of functional set of parameters and application program interface structures for each of the one or more plug-in processes.

8. The client computer of claim 7 wherein the plug-in interface process further comprises a core installer process that functionally couples the one or more plug-in programs together in accordance to the functional set of parameters.

9. The client computer of claim 8 wherein the core installer accesses a script process that specifies a network address of a script program that defines the functional set of parameters.

10. The client computer of claim 9 wherein the client interface process comprises one of a single instance core and a multiple instance plug-in core.

11. The client computer of claim 10 wherein the one or more plug-in programs include hidden plug-ins that are created in a hidden window by the plug-in interface process and are not specified on the web page.

12. The client computer of claim 10 wherein the one or more plug-in programs include native plug-ins that represent named objects that are specified on the web page.

13. The client computer of claim 9 wherein the web browser program comprises one of Netscape Navigator and Internet Explorer, and wherein each plug-in program of the one or more plug-in programs includes an interface allowing the plurality of parameters for each plug-in program to be passed to the web browser program.

14. A distributed computer network system comprising

a server computer executing a web server process providing World Wide Web content in Hypertext Markup Language format to one or more client computers coupled to the network system; and

a client computer executing a web browser process accessing web pages provided by the server computer, the client computer further including one or more plug-in processes executed in conjunction with the web browser process and operable to display content within a web page displayed on the client computer, and a plug-in interface process executed in conjunction with the web browser process and configured to define a common set of functional set of parameters and application program interface structures for each of the one or more plug-in processes.

15. The distributed computer network system of claim 14 wherein the plug-in interface process further comprises a core installer process that functionally couples the one or more plug-in programs together in accordance to the functional set of parameters.

16. The distributed computer network system of claim 15 wherein the core installer accesses a script process that specifies a network address of a script program that defines the functional set of parameters.

17. The distributed computer network system of claim 16 wherein the one or more plug-in programs include hidden plug-ins that are created in a hidden window by the plug-in interface process and are not specified on the web page, and native plug-ins that represent named objects that are specified on the web page.

18. The distributed computer network system of claim 17 wherein the web browser program comprises one of Netscape Navigator and Internet Explorer, and wherein each

plug-in program of the one or more plug-in programs includes an interface allowing the plurality of parameters for each plug-in program to be passed to the web browser program.

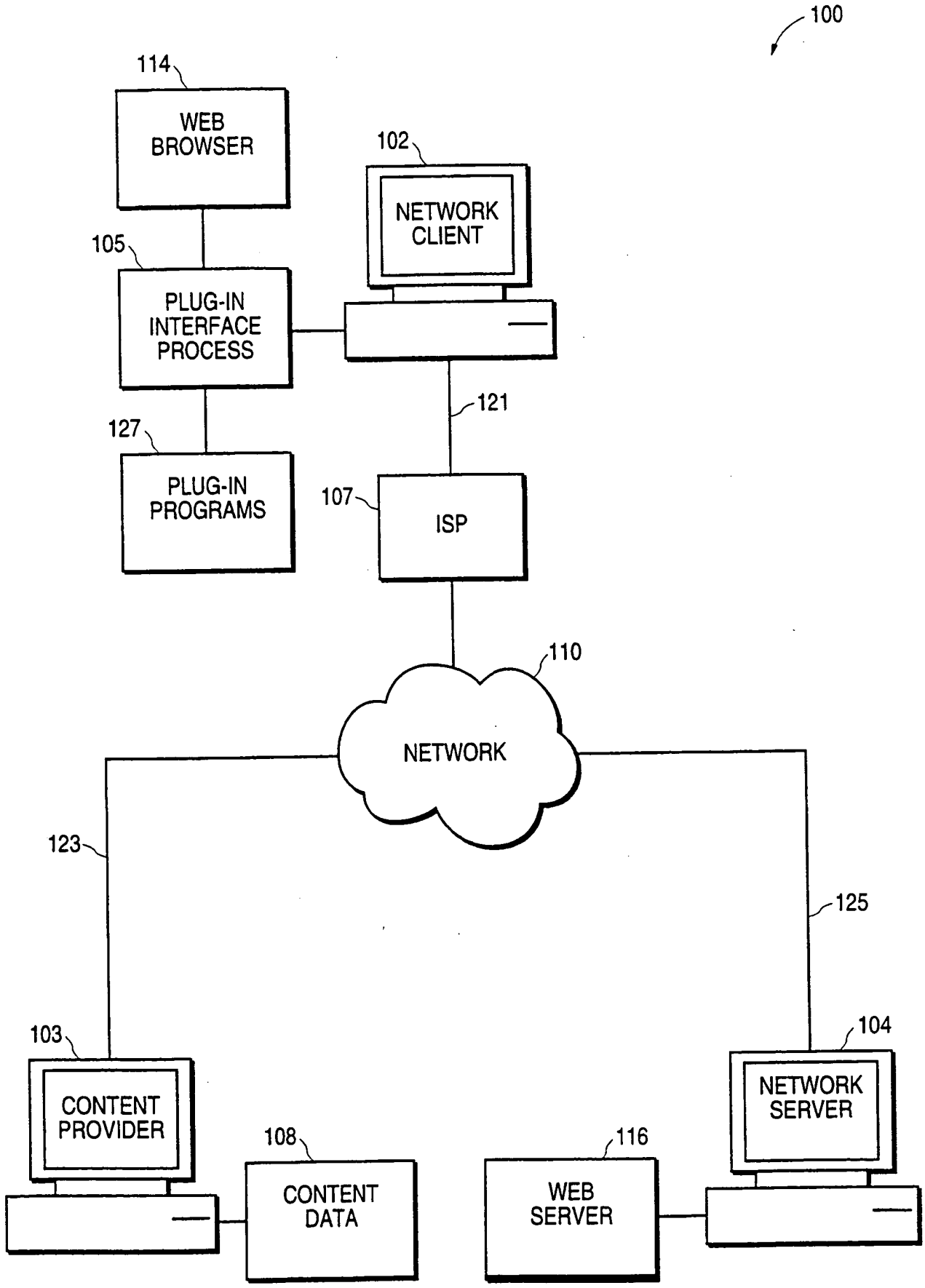

100
114
WEB BROWSER
102
NETWORK CLIENT
105
PLUG-IN INTERFACE PROCESS
121
127
PLUG-IN PROGRAMS
107
ISP
110
NETWORK
123
125
103
CONTENT PROVIDER
104
NETWORK SERVER
108
CONTENT DATA
116
WEB SERVER

FIG. 1

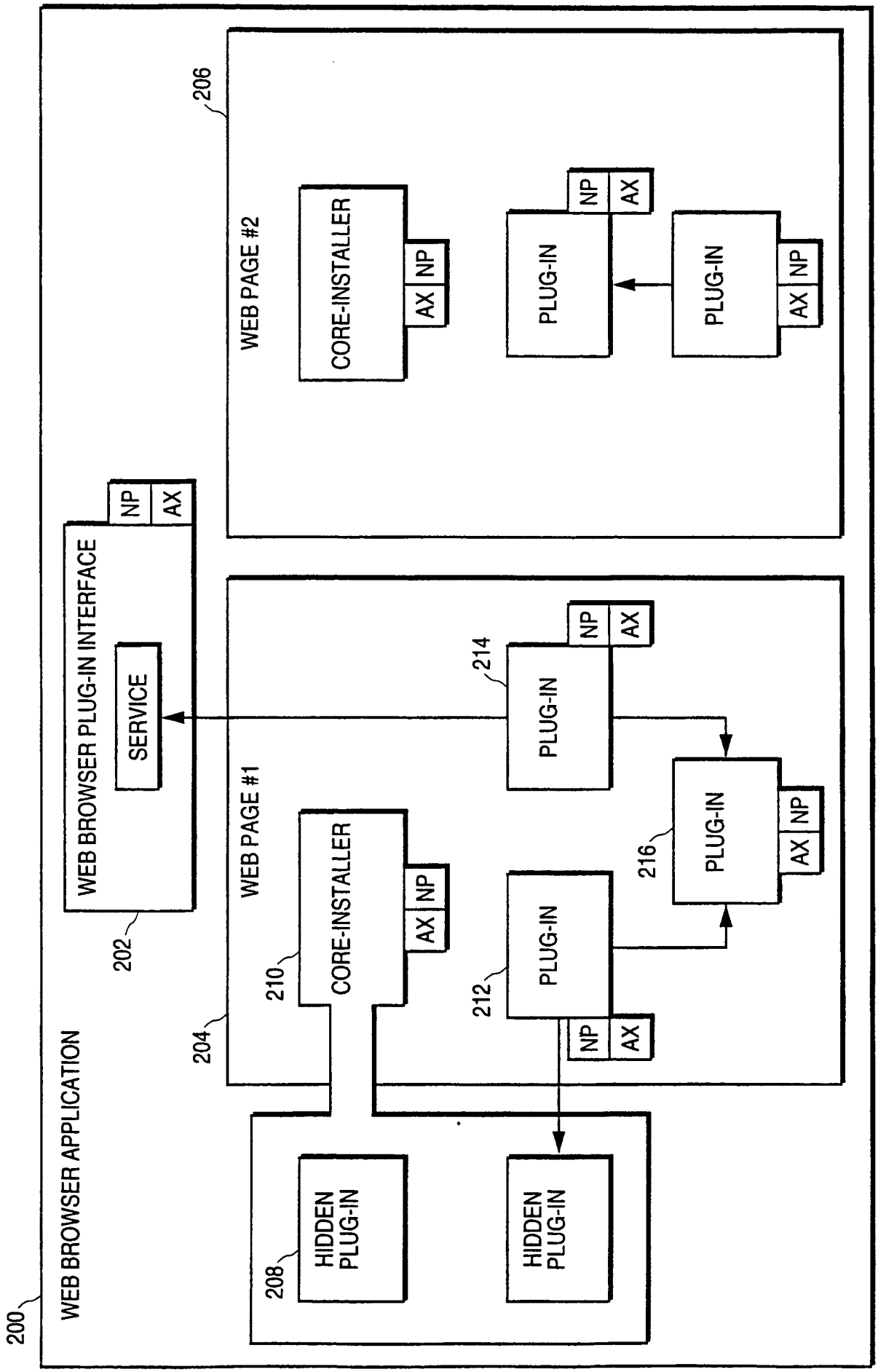

200
WEB BROWSER APPLICATION
202
WEB BROWSER PLUG-IN INTERFACE
NP
AX
SERVICE
204
WEB PAGE #1
210
CORE-INSTALLER
AX
NP
212
PLUG-IN
NP
AX
214
PLUG-IN
NP
AX
216
PLUG-IN
AX
NP
208
HIDDEN PLUG-IN
HIDDEN PLUG-IN
206
WEB PAGE #2
CORE-INSTALLER
AX
NP
PLUG-IN
NP
AX
PLUG-IN
AX
NP

FIG.2

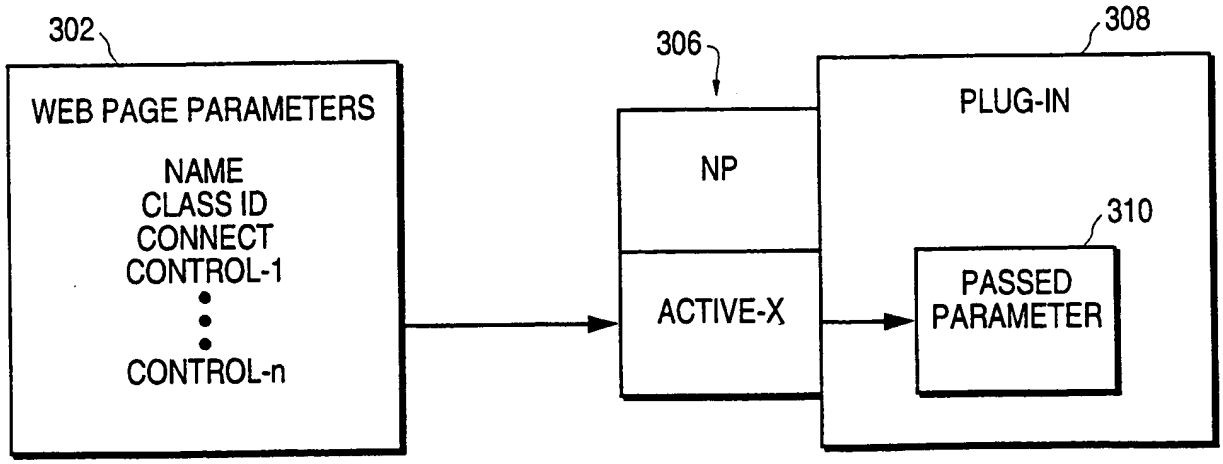

302
WEB PAGE PARAMETERS
NAME
CLASS ID
CONNECT
CONTROL-1
CONTROL-n
306
NP
ACTIVE-X
308
PLUG-IN
310
PASSED PARAMETER

FIG.3

INTERNATIONAL SEARCH REPORT

Int. ational Application No
PCT/US 00/32171

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
WPI Data, EPO-Internal, PAJ, INSPEC, IBM-TDB, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	"HOW PLUG-INS PLUG IN" WWW PLUG-INS COMPANION,XX,XX, 1996, pages 15-21, XP002911311 the whole document ---	1,7,14
A	LANCIE DE P: "WEB ANIMATION, AUDIO, AND VIDEO WITH EMBED" WEB TECHNIQUES,US,MILLER FREEMAN, April 1998 (1998-04), pages 65-67, XP000783668 ISSN: 1086-556X page 65, left-hand column, line 1 -page 67, left-hand column, line 6 --- -/--	1,7,14

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 March 2001	26/03/2001

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Deane, E

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Int .tional Application No
PCT/US 00/32171

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category ˘	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 958 008 A (WEINBERG AMIR ET AL) 28 September 1999 (1999-09-28) abstract; figure 12 column 17, line 40 -column 18, line 37; figure 7 -----	1,7,14

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int. .tional Application No
PCT/US 00/32171

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5958008 A	28-09-1999	US 5870559 A	09-02-1999
		US 6144962 A	07-11-2000
		US 5974572 A	26-10-1999